*fast*FM: A Library for Factorization Machines

Immanuel Bayer IMMANUEL.BAYER@UNI-KONSTANZ.DE
University of Konstanz
78457 Konstanz , Germany

Editor:

Abstract

Factorization Machines (FM) are only used in a narrow range of applications and are not part of the standard toolbox of machine learning models. This is a pity, because even though FMs are recognized as being very successful for recommender system type applications they are a general model to deal with sparse and high dimensional features. Our Factorization Machine implementation provides easy access to many solvers and supports regression, classification and ranking tasks. Such an implementation simplifies the use of FM's for a wide field of applications. This implementation has the potential to improve our understanding of the FM model and drive new development.
Keywords: Matrix Factorization, Recommender Systems, MCMC

1. Introduction

This work aims to facilitate research for matrix factorization based machine learning models. Factorization Machines are able to express many different latent factor models that are widely used for collaborative filtering tasks Rendle (2012b). An important advantage of FM's is that side information can be easily included.

$$w_0 \in \mathbb{R}, x, w \in \mathbb{R}^p, v_i \in \mathbb{R}^k$$

$$\hat{y}^{FM}(x) := w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j>i}^{p} \langle v_i, v_j \rangle x_i x_j \tag{1}$$

For example, encoding a sample as $x = \{\cdots, \overbrace{1}^{x_i}, \cdots, \overbrace{1}^{x_j}, \cdots\}$ yields $\hat{y}^{FM}(x) = w_0 + w_i + w_j + v_i^T v_j$ which is equivalent to (biased) matrix factorization $R_{i,j} \approx b_0 + b_i + b_j + u_i^T v_j$ Srebro et al. (2004). Please refer to Rendle (2012b) for more encoding examples. FM's are, however, more flexible than standard matrix factorization by learning all higher order interactions between features (implemented up to second order [eq. 1]). It has been the top performing model in machine learning competitions Rendle and Schmidt-Thieme (2009), Rendle (2012a), Bayer and Rendle (2013) with different objectives (e.g. What Do You Know? Challenge[1], EMI Music Hackathon[2]). This library includes solvers for regression, classification and ranking problems (see table 2) and addresses the following needs of the research community:

1. `http://www.kaggle.com/c/WhatDoYouKnow`
2. `http://www.kaggle.com/c/MusicHackathon`

(i) Easy interfacing for dynamic and interactive languages such as R, Python and Matlab. (ii) A Python interface that allows interactive work. (iii) A publicly available testsuite that strongly simplifies modifications or adding of new features. (iv) Code is released under the **BSD-license** which allows the integration in (almost) any open source project.

2. Design Overview

The *fast*FM library has a multi layered software architecture (table 1) that clearly separates the interface code from the performance critical parts (*fast*FM-core). The core contains the solvers, is written in C and can be used stand alone. Two user interfaces are available, a command line interface (CLI) and a Python interface. They serve as reference implementation for bindings to additional languages.

2.1 fastFM-core

*fast*FM (Py)	
Cython	CLI
*fast*FM-core (C)	

Table 1: Library Architecture

The strength of FM's is the ability to perform well on problems with very high dimensional categorical features as they frequently occur for example in click through rate prediction. This often leads to design matrices with more then 95% sparsity. We use the standard compressed row storage (CRS) matrix format as underlying data structure and rely on CXSparse[3] for fast sparse matrix / vector operations Davis (2006). This simplifies the code and makes memory sharing between Python and C easy, which is critical as memory copies would reduce performance for important operations.

A test suite that provides high test coverage and routines that check that the code is Valgrind-clean ensures that the code is easy to extend. Solvers are tested using state of the art techniques such as Posterior Quantiles (Cook et al., 2006) for the MCMC sampler or Finite Differences for SGD based solvers.

2.2 Solver and Loss Functions

Table 2 lists the supported tasks and the solvers available for each. The MCMC solver implements the Bayesian Factorization Machine model (Freudenthaler et al.) via Gibbs sampling. We use the Bayesian Personalized Ranking (BPR) pairwise loss (Rendle et al., 2009) for ranking. More details on the classification and regression solver can be found in Rendle (2012b).

Task	**Solver**	**Loss**
Regression	ALS, MCMC, SGD	Square Loss
Classification	ALS, MCMC, SGD	Probit (MAP), Probit, Sigmoid
Ranking	SGD	BPR Rendle et al. (2009)

Table 2: Supported Solvers and Tasks.

3. CXSparse is LGPL licensed.

2.3 Python Interface

The Python Interface is compatible to the API of the widely-used SCIKIT-LEARN library Pedregosa et al. (2011) which opens the library to a large user base. The following code snippet shows how to use MCMC sampling for a FM classifier and make predictions on new data.

```
fm = mcmc.FMClassification(init_std=0.01, rank=8)
y_pred = fm.fit_predict(X_train, y_train, X_test)
```

Additional features are available such as warm starting a solver from a previous solution.

```
fm = als.FMRegression(init_std=0.01, rank=8, l2_reg=2)
fm.fit(X_train, y_train)
fm.fit(X_train, y_train, n_more_iter=10)
```

3. Experiments

We first show that the ALS and MCMC solver are comparable with libFM[4] with respect to runtime and convergence (figure 1). A fixed number of 50 iterations is used for all comparisons. The x-axis indicates the number of latent factors (rank) and the upper plots compare the implementations with respect to decrease in rmse error. The lower plots show how the runtime scales with the rank. The figure indicates that *fast*FM has competitive performance, with both the CLI and the high-level Python interface. Figure (2) compares MCMC chain traces using warm-start (dashed) and retraining the model (solid) line. Many other analysis are simplified by interactive access to solvers and model parameter. The experimental details behind figures 1 and 2 can be found in the on-line documentation[5].

4. Related Work

Factorization Machines are available in the large scale machine learning libraries GraphLab (Low et al., 2014) and Bidmach (Canny and Zhao, 2013). The toolkit Svdfeatures by Chen et al. (2012) provides a general MF model that is similar to a FM. The implementations in GraphLab, Bidmach and Svdfeatures only support SGD solvers and can't be used for ranking problems. libFM contains all the solvers available in our *fast*FM implementation but lack's an interactive user interface and a publicly available test-suit. *fast*FM's main objective is to be easy to use and extend without sacrificing performance.

Acknowledgments

This work was supported by the DFG under grant Re 3311/2-1.

4. `http://libfm.org`
5. Source code and on-line documentation is available at `https://github.com/ibayer/fastFM`.

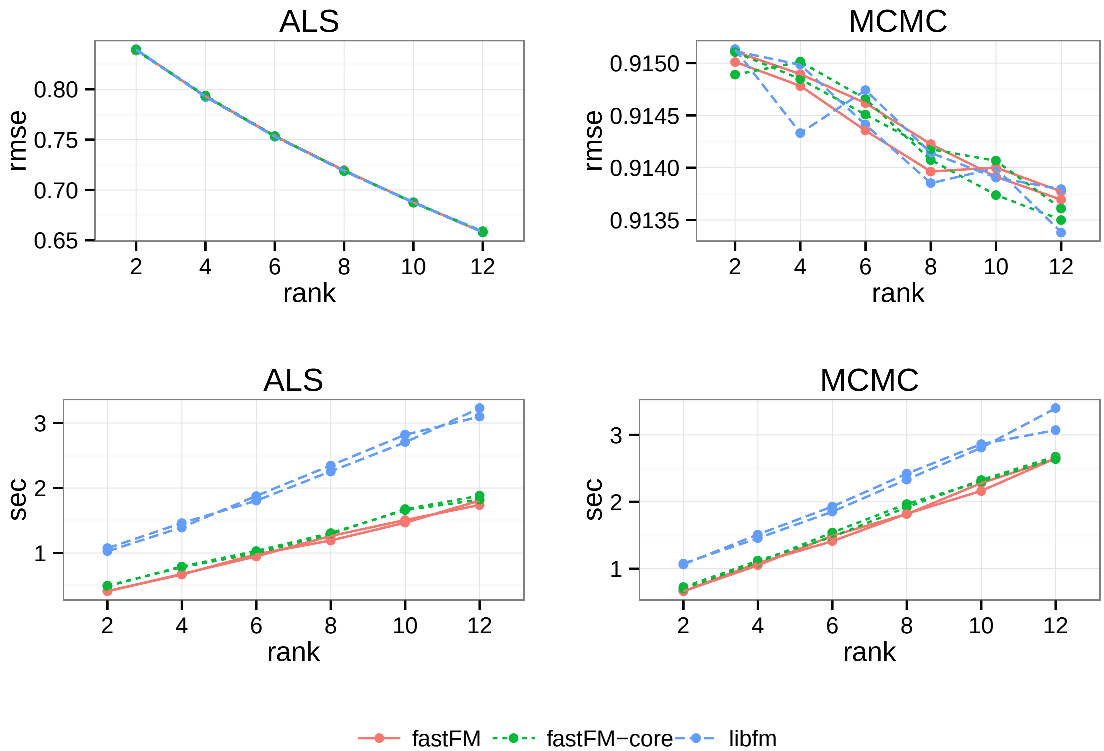

Figure 1: Runtime and convergence comparison between the two *fast*FM interfaces and libFM. The evaluation is done on the movielens ml-100-k dataset and train error is reported.

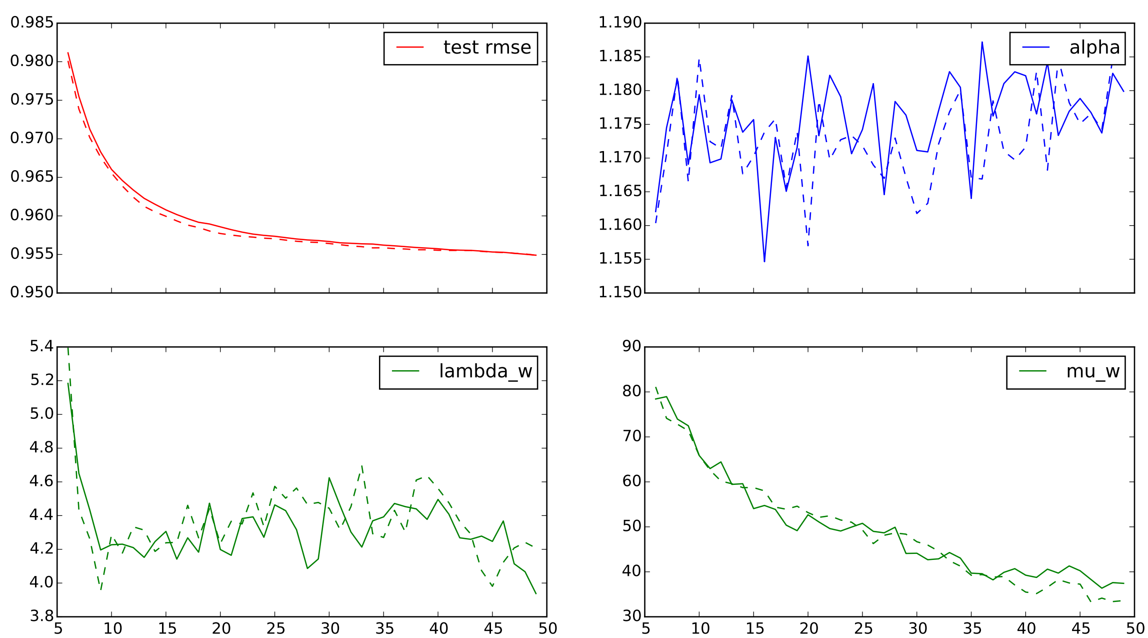

Figure 2: MCMC chain analysis and convergence diagnostics example.

References

Immanuel Bayer and Steffen Rendle. Factor models for recommending given names. *ECML PKDD Discovery Challenge*, page 81, 2013.

John Canny and Huasha Zhao. Bidmach: Large-scale learning with zero memory allocation. In *BigLearn Workshop, NIPS*, 2013.

Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *The Journal of Machine Learning Research*, 13(1):3619–3622, 2012.

Samantha R Cook, Andrew Gelman, and Donald B Rubin. Validation of software for bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics*, 15(3), 2006.

Timothy A Davis. *Direct methods for sparse linear systems*, volume 2. Siam, 2006.

Christoph Freudenthaler, Lars Schmidt-thieme, and Steffen Rendle. Bayesian factorization machines.

Yucheng Low, Joseph E Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E Guestrin, and Joseph Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041*, 2014.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Steffen Rendle. Social network and click-through prediction with factorization machines. 2012a.

Steffen Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3 (3):57:1–57:22, May 2012b. ISSN 2157-6904.

Steffen Rendle and Lars Schmidt-Thieme. Factor models for tag recommendation in bibsonomy. In *ECML/PKDD 2008 Discovery Challenge Workshop, part of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 235–243, 2009.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. UAI '09, pages 452–461, Arlington, Virginia, United States, 2009.

Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. Maximum-margin matrix factorization. In *Advances in neural information processing systems*, pages 1329–1336, 2004.